Assignment 2

Start:	10 October 2011
End:	24 October 2011

Objectives

In this assignment you will learn to develop distributed Web applications using the two different paradigms that you have seen in the lecture: *REST* and *WS-**. You will make use of these paradigms when implementing a mobile phone application that gathers data from services provided over the Web by wireless sensor nodes (SunSPOTs[1]). The SunSPOTs expose their services (e.g., temperature sensor, ambient light sensor, etc.) through two different interfaces: REST-based and WS*-based (cf. Figure 1).

- **Representational State Transfer (REST)** is a style of software architecture for implementing Resource Oriented Architectures (ROAs). HTTP (1.1)[2], the application protocol of the Web, represents an implementation of the REST principles. Distributed RESTful applications can be developed using the HTTP protocol as a universal interface for interacting with resources on the Web. Such applications make use of HTTP verbs (GET, POST, PUT, DELETE, etc.) and mechanisms (e.g., URIs, HTTP Content Negotiation). Furthermore, REST defines how to serve different formats (e.g., HTML, JSON, XML) for a given resource depending on the clients needs.
- **WS-* services**, sometimes called "Big Web services", describe a set of XML-based standards (e.g., WSDL, SOAP, UDDI) that can be used to implement Service Oriented Architectures (SOAs). Rather than using HTTP as an application protocol, WS-* services use it as a transport protocol and define a number of additional layers to encapsulate distributed services.

With this assignment you can gain 10 points out of the total 45. Regarding the naming of your Eclipse projects, use *VS_G**_A2_{TaskNumber}* and *ch.ethz.inf.vs.android.g**.a2* as project/package names.

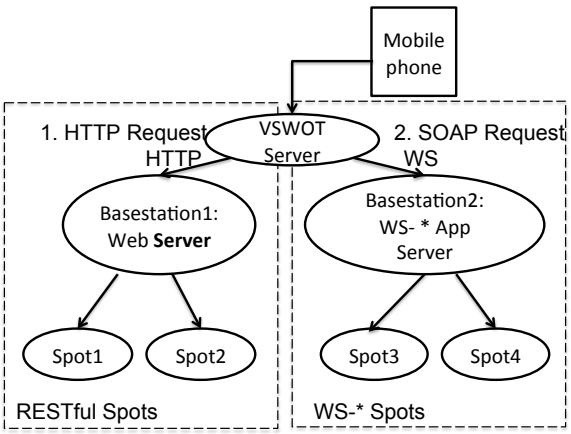

Figure 1: System setup for Assignment 2: Sensor nodes can be accessed either directly through HTTP or using SOAP messages.

[1] http://sunspotworld.com/
[2] http://www.w3.org/Protocols/rfc2616/rfc2616.html

1 Experimenting with RESTful Web Services (2 Points, ⊙)

The SunSPOTs *Spot1* and *Spot2* deliver the sensor values through a RESTful interface. Open your browser and navigate to: `http://vswot.inf.ethz.ch:8081/sunspots/`. From there you can explore the HTML representation of the RESTful SunSPOTs. Browse and experiment with the SunSPOTs *Spot1* and *Spot2* and look at the sensor values they offer through this Web interface (e.g., temperature, light, acceleration). You can also use the Spots' actuators to switch on or off their LEDs and choose colors. Your task is to write an Android application that requests the temperature measurement from *Spot1* and displays the value on the screen. Implement each subtask in a new function and provide an additional button to perform the action.

What to do

- RESTful Web services can be invoked using the HTTP protocol. In this assignment, we start by making a "raw" HTTP request *without the use of any external library*. Open a TCP connection to *vswot.inf.ethz.ch* (port 8081) and make an HTTP GET request to obtain the temperature information of *Spot1*. Display the raw response on the screen.
  Hints:
  - Use the `Socket` class in the `java.net` library. To send and receive data use the `getInputStream()` and `getOutputStream()` methods of the `Socket` class and read and write to the corresponding `InputStream` and `OutputStream`.
  - For the HTTP protocol have a look at `http://www.elektronik-kompendium.de/sites/net/0902231.htm` or google for other howtos.
  - Take care to correctly implement the HTTP protocol: HTTP headers are required to have a *carriage return and newline* at the end of each line. *println(...)*, however, uses the *line.separator* property, which is just a *newline* for Android. Also, don't forget to *flush* when using *PrintWriter*.
- Several different libraries can be used to create HTTP calls. Use the Apache HTTP Client library (`import org.apache.http.*` or another HTTP Client/REST library of your choice) for sending the HTTP request to get the temperature resource from *Spot1*. Display the raw response on the screen.
- So far, we only got HTML responses back from the SunSPOTs. As mentioned before, a RESTful service can offer several representations of the same resource. To get a different representation that is more appropriate for machine-machine communication than HTML, we use the HTTP Content Negotiation mechanism: Set the `Accept` header of your HTTP request to `application/json`. Instead of an HTML page, the Web server will now return a JSON file containing the temperature information. JSON is a lightweight version of XML which is often used in Web mashups and RESTful interfaces because it can directly be translated to JavaScript objects. Display the raw JSON response.
- Parse the retrieved JSON response to extract the temperature value. Display only this value on the screen.
  Hints: You can find several libraries on the Web to parse JSON for many languages[3]. Android already includes `org.json.*` so you can directly instantiate and work with these JSON classes (i.e., `org.json.JSONObject`).

[3] `http://json.org`

2 Experimenting with WS-* Web Services (2 Points, ⊙)

While *Spot1* and *Spot2* offer a RESTful interface, *Spot3* and *Spot4* can be accessed using their WS-* Web Service interface. Open your browser and navigating to: `http://vswot.inf.ethz.ch:8080/SunSPOTWebServices/SunSPOTWebservice`. From there you can access the WSDL (Web Services Description Language) description of the offered functionality. Have a closer look at the WSDL interface and try to understand its content and what it provides as you did for the RESTful version. Finally, use the HTML interface at `http://vswot.inf.ethz.ch:8080/SunSPOTWebServices/SunSPOTWebservice?Tester` to test the Web Service from your browser.
Hint: To fully understand the WSDL description, also look at the *schema* specified in the WSDL file.

What to do

- WS-* Web services can be invoked by first getting their WSDL and then sending SOAP messages to the Web service endpoint. A WSDL contains a description of these SOAP messages and end-points. Describe step-by-step how you would proceed to invoke a Web service using the `java.net` library only, as you did for RESTful Web Services. Include this description in your report. You do not need to implement anything for this task.

- Create a **new** application and use a WS-* library (e.g., the kSOAP2 library patched to work on Android[4] or another WS-* library of your choice) to make WS-* calls from the Android device. Implement a button to get the temperature data from *Spot3*. Also display the raw SOAP response on the screen.
  Hints:

  - Check out `http://code.google.com/p/ksoap2-android/wiki/Links`.
  - To display the raw SOAP response, you will have to enable the *debug* mode in the *org.ksoap2.transport.AndroidHttpTransport*

  Hint:

- When using WS-* services, messages are represented using XML "over the wire" and then unmarshalled on the client side into platform specific objects (SOAPObjects in the case of kSOAP2). Find out how to get the XML containing the temperature and implement a button to display it as raw XML on your Android phone.

- Parse the retrieved SOAPObject (or XML response) to extract the temperature value. Implement a button to display this value on the screen. Feel free to use a library to parse the value.

3 Assessing Web Service Technologies (1 Point)

Evaluate the two technologies, REST and WS-*, for working with embedded devices.

What to do

- Answer the questions at `https://docs.google.com/spreadsheet/viewform?hl=en_US&formkey=dDlPSFRuV1BocjNCTlA0d1FpMERGRlE6MA#gid=0`. Do not forget to state your group number and nethz login name of the group leader at the beginning of the form.

[4] `http://code.google.com/p/ksoap2-android/`

4 Cloud Services (1 Point)

In this task, you should visualize the data that you retrieved from the sensor nodes using both, local visualization and an external service to provide the graphics. To access the sensors, you may choose between REST and WS-* APIs.

What to do

- Use the native Android graphics libraries to visualize the retrieved data using, e.g., a graph that shows the temperature trend over some minutes. Consider e.g., `http://marakana.com/tutorials/android/2d-graphics-example.html` or similar tutorials to get started.
- Use the Google Chart API to display the sensor measurements you retrieved with your phone. You can read more about the specific display methods at: `http://code.google.com/apis/chart/interactive/docs/gallery.html`. Pick your favorite visualization scheme. Send the measured values to the service and display the visualization image on the phone's screen.

5 Your Phone as a Server (2 Points)

Similar to the SunSPOT sensor node, your phone can also provide RESTful Web Services for its sensing and actuation functionality. Your task is to make two sensors and two actuators through a REST server running on your mobile phone.

What to do

- Implement the server in a similar manner to the `Socket` client in Task 1, only now make use of `ServerSocket`. Wait for and `accept` incoming connections and then handle the requests according to the HTTP protocol. You will have to pick a port greater than 1024 (e.g., 8081).
  Hints:
  - Reuse your implementation to access sensors and actuators from Assignment 1, Task 1.
  - Your server does not have to be RFC-compliant. Implement a minimal version that works with your browser (e.g., header options can be ignored)
- Run the Android 2.2 Wi-Fi Hotspot app and connect your laptop to the phone. Use your laptop's browser to test the functionality of your Android Web server.
- Add multi-threading to your Web server: The server should be able to accept and answer requests form several clients at once.
- Describe your chosen architecture and control flow to fulfill client requests in the report, even if you might not be able to get the server running.

6 Report (2 Points, ⊙)

As part of the assignments, you should produce a short report **(1-2 pages)**. Please write about the design and implementation questions of your applications and motivate any choices you have made during the process. Indicate any problems you have encountered during the development. You can include code snippets to explain particular ideas and we encourage you to highlight bits you are especially proud of (or don't like at all). Feel free to also compare the Android platform to other devices such as Symbian or

iOS if you have previously gained experience in those. If you provide a solution for the final part of this assignment, we expect you to introduce your enhancements and evaluate their usefulness. A template for your reports will be given on the course website.

Deliverables

The following deliverables have to be submitted by **9:00am, 24 October 2011**:

1. **code.zip** You should create a zip file containing the Eclipse projects created in this assignment. The projects should have been tested both on the mobile phone and on the emulator. The code must compile on our machines as well, so always use relative paths if you add external libraries to your project. Do not forget to include those libraries in the zip file. Please use UTF-8 encoding for your documents and avoid special characters like umlauts.
2. **report.pdf** The report in **pdf** format.
3. Technology-related questions for **task 3** at: `https://docs.google.com/spreadsheet/viewform?hl=en_US&formkey=dDlPSFRuV1BocjNCTlA0d1FpMERGRlE6MA#gid=0`.
4. **feedback form** Tell us your experience with learning the REST and WS-* technologies (anonymous & individually) at: `https://docs.google.com/spreadsheet/viewform?formkey=dFFsbS1OVUVSaV9Id1dUYjZ1N0Jsdmc6MA#gid=0`.

Submission

Code and report must be uploaded through:

`https://www.vs.inf.ethz.ch/edu/vs/submissions/`

The group leader can upload the files, and other group members have to verify in the online system that they agree with the submission. Use your nethz accounts to log in. The submission script will not allow you to submit any part of this exercise after the deadline. However, you can re-submit as many times as you like until that.